What is Content Projection

- Re-use of content *inside* of components
- Often used when you create components *to be used* by others

- Simple use:
  - *Attribute binding* to pass data into components `[prop]="data"`
  - *Event binding* to get data out of components

    `(event)="handler()"`

Examples ../130-content-projection

Content Projection

localhost:4200

Component: Card 1

Relax

Get some rest while visiting beautiful landscapes and enjoying the mountains and the sea.

No content projection used in this component

LEARN MORE

Component: Card 2

Free life

Escape the city, avoid rush hour, no more deadlines. Get away and be free.

The title of this card is projected

LEARN MORE

Component: Card 3

Enjoy nature

Enjoy the outdoor life. Fly like an eagle and get rid of your anger, worries and stress.

The title of this card and the text contents are projected. We are using class projection here.

LEARN MORE

Component: Card 4

Hike your path

Component: Card 5

1. Simple content projection:

```
<my-component>
    I want to re-use this text
</my-component>
```

In the parent component:

```
<app-card2>
   Free life
</app-card2>
```

In the child component:

```
<h4 class="card-title">
  <ng-content></ng-content>
</h4>
```

2. Content projection based on CSS-class

```
In the parent component:

<app-card3>

<p class="card-text">Enjoy...</p>

</app-card3>
```

```
In the child component:

<ng-content
    select=".card-text">

</ng-content>
```

3. Content projection based element selector

```
In the parent component:

<app-card4>
   <img class="card-photo"
      src="assets/img/photo-04.jpg">
    …

</app-card4>
```

```
In the child component:

<ng-content
   select="img.card-photo">
</ng-content>
```

4. Based on custom component

- Create an extra component (here: `<app-newsletter>`)
- Use content projection based on element selector
- Extra: submit events from nested component back to parent

```
<div>
   <label>
      <input type="checkbox" (change)="onChange($event.target.checked)">
      Subscribe to newsletter!
   </label>
</div>
```

```
export class NewsletterComponent {

   @Output() checked: EventEmitter<boolean> = new EventEmitter<boolean>();

   onChange(value: boolean) {
      this.checked.emit(value)
   }
}
```

```
In parent Component:

<app-card5>
   …
   <app-newsletter (checked)="onChecked($event)">
   </app-newsletter>
   …
</app-card5>

…handle event on parent component class
```

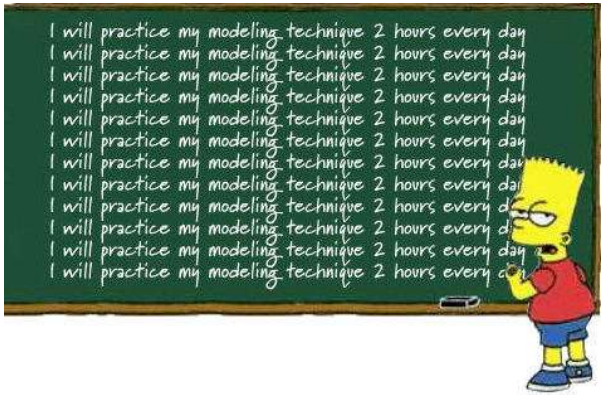
```
<!-- nested component, projected from parent component -->
<ng-content select="app-newsletter"></ng-content>
```

Verdict

- Use Content Projection, to present VIEW information inside the component
  - Again: *mostly* used on redistributable components
- Use `@Input()` and `@Output()` decorators for logic of the component

Workshop

- Create a new, custom button component (`<my-button>`)
- The contents of that button are:
  - General attributes: background red, 180x65px, 2px border solid black
  - An icon (save, login, profile, etc)
  - Text (‘save’, ‘login’, ‘profile’, etc)
- Each component instance should have the same general layout, but different contents that is content projected inside the button component
  - Text
  - Icon



Workshop

- Open `../130-content-projection` for examples, or use your own app
- Create a new component
- Use `<ng-content>` to project content from outside into the component.
- Add an `<ng-content>` class selector. Use it from the outside component.
- Add an `<ng-content>` element selector. Use it from the outside component.
- Create another component and nest it inside your child component. Use the element selector to project it. Optional: Propagate events up